Enhancing Local Search with Adaptive Operator Ordering and Its Application to the Time Dependent Orienteering Problem

Aldy Gunawan · **Hoong Chuin Lau** · **Kun Lu**

Abstract In most metaheuristics, such as Iterated Local Search (ILS), Simulated Annealing (SA) and Tabu Search (TS), the Local Search (LS) procedure is embedded in order to generate possible neighborhood solutions. LS consists of several operators, such as SWAP, 2-OPT and others, that would be called either independently or sequentially. In this paper, we deal with the extension of Adaptive Operator Selection (AOS). Instead of calling one operator at each iteration, we focus on how to arrange the sequence of operators at one particular iteration. We introduce two metaheuristics which are based on ILS and a hybridization of SA and ILS (SAILS), namely Adaptive ILS and Adaptive SAILS. In these two metaheuristics, we include a mechanism based on the reinforcement learning system, called Learning Automata, to calculate the probability of selecting the operators. Based on these probabilities, we generate a sequence of operators for the next iteration by proposing three different scenarios: rank-based selection, random selection and fitness proportionate selection. Adaptive ILS and Adaptive SAILS are implemented to solve a variant of the Orienteering Problem (OP), namely Time Dependent OP. The extensive computational results show the superiority of Adaptive SAILS with the fitness proportionate selection. We also conclude that the adaptive learning mechanism outperforms the state-of-the-art algorithm for some benchmark instances.

Keywords Local Search · Adaptive operator selection · Time Dependent Orienteering Problem · Iterated Local Search

1 Introduction

Metaheuristics are generic frameworks for solving optimization problems in contrast with problem-specific heuristics [2]. Some examples of metaheuristics are Iterated Local Search (ILS), Simulated Annealing (SA), Tabu Search (TS) and Ant Colony

A. Gunawan, H.C. Lau and K. Lu
Singapore Management University
E-mail: aldygunawan, hclau, kunlu@smu.edu.sg

Optimization (ACO). Metaheuristics are effective in solving many combinatorial optimization problems with small computational budget.

A Local Search (LS) procedure is often the underlying mechanism in metaheuristics for generating a set of possible neighborhoods [13]. Central to LS are a set of operators, such as SWAP, 2-OPT and others. The quality of operators influences the performance of LS. The fundamental question in the design of LS is how to arrange the sequence of operators at each iteration step. In most cases, the sequence is static and deterministic, that is to say that the sequence is fixed at the beginning and constant throughout the run of the algorithm. However, a number of authors have proposed methods of adaptively controlling the operators [16].

Adaptive Operator Selection (AOS) is an online adaptive algorithm that adjusts the probability of applying operators and select one suitable operator of Local Search (LS) to the current solutions [12]. Thierens [18] classified adaptation methods in two classes: self-adaptation and adaptive allocation rule. The former directly encodes the values of the operator probabilities in the representation of the individual solutions while the latter adapts the values of the operator probabilities based on an learning rule.

AOS has been used quite extensively to solve complex optimization problems. Burke et al. [3] applied it in the context of hyperheuristics for solving challenging optimization problems. Thierens [19] introduced the adaptive pursuit algorithm for selecting the perturbation step size of ILS. Computational results suggest that the adaptive pursuit algorithm is able to achieve almost the same performance as the ILS with the best perturbation step size, without the need to determine the optimal parameter setting, in solving a knapsack problem.

Burke et al. [4] proposed two adaptive variants of a multiple neighborhood ILS. Online learning techniques are employed in order to select which perturbation to apply iteratively from a set of available move operators. The proposed algorithms were tested on four different combinatorial optimisation problem: permutation flow shop, one-dimensional bin packing, maximum satisfiability and personal scheduling problems. Experimental results suggest that the adaptive variants outperform a baseline ILS with uniform random selection of the move operators. Yuan et al. [23] proposed an online parameter adaptation on the operator selection problem in Memetic Algorithm (MA) for solving the Quadratic Assignment Problem (QAP). The performance of online AOS methods is improved by considering different reward functions. Soria-Alcaraz et al. [17] dealt with AOS in the context of the evolutionary algorithms. Their proposed idea is to use metrics based on local characteristics of the fitness landscape surrounding a solution to measure the impact of operators. They applied this idea to three problems, Onemax, Royal Staircase and Multiple Knapsack Problems.

In this paper, we focus two well known metaheuristics, ILS and SA, for solving a variant of the Orienteering Problem (OP), namely the Time Dependent OP (TDOP). For other methods and other variants of the OP, we refer the readers to the survey of Vansteenwegen et al. [20] and Gunawan et al. [9]. OP is a routing problem in which the goal is to determine a subset of nodes to visit within a particular path, and in which order, so that the total collected score is maximized and a given time budget is not exceeded. In TDOP, the travel time between two nodes depends on the departure time at the first node. In this problem, the number of paths is set to 1.

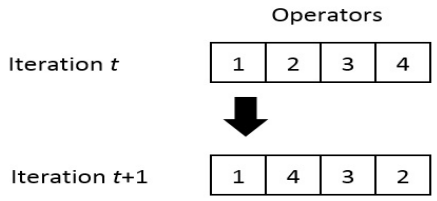

Fig. 1: Adaptive Operator Ordering

One common example is the combination of walking and using public transport that affects the travel time for tourist trip planners [6]. TDOP is NP-hard since OP itself is NP-hard [5].

Abbaspour and Samadzadegan [1] introduced two adaptive genetic algorithms for solving the TDOP in the context of the city of Tehran. In these algorithms, chromosomes with variable lengths are used. Verbeeck et al. [21] introduced a fast solution algorithm based on an Ant Colony System with a time dependent LS procedure. Realistic benchmark instances are also constructed.

We extend the OP literature on AOS. Instead of calling and applying operators individually at each iteration, we focus on how to determine the sequence of LS operators. Some dependencies among operators may occur and multiple operators may provide better results through interactions. It is called Adaptive Operator Ordering (AOO). This idea can be treated as an adaptive RELAY HYBRIDISATION approach [14] which determines a pair of low-level heuristics that can be used consecutively. In our problem, we extend this by considering more than two operators. Here, we treat operators of LS as low-level heuristics.

Figure 1 shows one example how the sequence of operators for two different iterations are generated. At iteration t, Operator 1 would be called first, followed by Operators 2, 3 and 4. After applying a particular scenario for updating the sequence of the operators, the sequence would be rearranged at iteration $(t+1)$.

A similar idea has been proposed by Walker et al. [22] in the context of hyper-heuristics. HyFlex, a software framework, is proposed to assist in hyper-heuristics and autonomous search control. They focus on selecting the move operators in the perturbation stage of the adaptive ILS. Another variant considers the past performance of the local search heuristics and put them in a sequence based on their performance is also proposed. Those heuristics are then applied based on this order (rank-based scenario). The proposed algorithms are tested on the vehicle routing problem with time windows. By adding adaptation mechanisms, the performance of hyper-heuristics is further improved.

We introduce two algorithms, namely Adaptive ILS (ADILS) and Adaptive SAILS (ADSAILS) by incorporating an adaptive learning method based on Learning Automata. Both are the extended versions of ILS [7] and SAILS (a hybridization of SA and ILS) [8], respectively. A learning automaton is an adaptive decision making unit that improves its performance by learning how to choose the action from a finite set of allowed actions through repeated interactions with a random environment [15]. In the context of our problem, actions are equivalent to operators of LS. The operator

Algorithm 1 ITERATED LOCAL SEARCH (ILS)

```
Set t ← 0 // iteration 0
Generate an initial solution S_0
S* ← LOCAL SEARCH (S_0)
repeat
    t ← t + 1
    S' ← PERTURBATION (S*, history)
    S" ← = LOCAL SEARCH (S') // iteration t
    S* ← = ACCEPTANCE CRITERION (S*,S", history)
until termination condition is met
```

is chosen based on a probability distribution kept over the operator-set. The operator probability values depend on the reinforcement feedback from the environment. With this selection, at each iteration step, the probabilities of selecting successful operators are increased, while the ones of unsuccessful operators would be decreased.

In order to generate a sequence of operators, we compare three different scenarios: 1) random selection, 2) rank-based selection and 3) fitness proportionate selection (Roulette-Wheel selection). The performance of LA is measured through the experiment on benchmark TDOP instances. Further comparisons with the state-of-the-art algorithm with fixed sequence of operators are also conducted. We conclude that ADSAILS with the fitness proportionate selection performs best. The obtained results also show the superiority of LA over other algorithms in terms of the overall average of best and average deviation values from optimal solutions.

This paper is organized as follows. Section 2 introduces three proposed scenarios: random selection, rank-based selection and fitness proportionate selection, for ordering the operators of LS. Section 3 provides the descriptions of benchmark instances, experimental set-up and comprehensive computational results. Finally, Section 4 gives concluding remarks and suggestions for future work.

2 Adaptive Operator Ordering

As mentioned earlier, our ADILS and ADSAILS are adopted from ILS [7] and SAILS [8], respectively. Take note that ILS and SAILS use a fixed order of the operators in their LOCAL SEARCH. In this work, we emphasize on how to call the sequence of operators for both ADILS and ADSAILS.

ILS [7] is a simple yet powerful metaheuristic that contains of two basic components, LOCAL SEARCH and PERTURBATION, for generating new solutions. Operators included in LOCAL SEARCH are as follows: SWAP, 2-OPT, REPLACE and INSERT. First, we generate the initial solution S_0 and apply LOCAL SEARCH to reach the current solution S^*. Given S^*, we apply PERTURBATION that leads us to an intermediate solution S'. LOCAL SEARCH is then applied to S' to reach a solution S". If S" passes ACCEPTANCE CRITERION, it becomes S^*; otherwise, we return to S^*. Algorithm 1 summarizes the basic idea of ILS.

SAILS [8] is a hybrid algorithm that combines ILS and Simulated Annealing (SA). The major difference between ILS and SAILS lies on ACCEPTANCE CRITERION. Unlike ILS, SA may accept a worse solution S" with a probability that changes

over time in order to escape from a local optimum. For more details, we refer to the original papers [7, 8].

We propose an adaptive learning method based on Learning Automata (LA) in order to calculate the probability of selecting the operators for the next iteration. In our context, each iteration refers to the step of calling LOCAL SEARCH (Algorithm 1). The probability value of each operator depends on the performance quality after applying the operator. In the context of the TDOP, a successful operator is able to either improve the objective function value or increase the remaining time budget. The sequence of operators thus relies on those probability values. In the context of hyperheuristics, selecting the next operator can be linked to the choice function of hyperheuristics [3]. In the following subsections, we describe three different scenarios for generating the sequence of operators: 1) random selection, 2) rank-based selection, and 3) fitness proportionate selection (Roulette-Wheel selection).

2.1 Random Selection

We have a set of K operators $A = \{a_1, \ldots, a_K\}$ and a probability vector at iteration t, $P(t) = \{P_{a_1}(t), \ldots, P_{a_K}(t)\} (\forall a_i \in A : 0 \leq P_{a_i}(t) \leq 1; \sum_{i=1}^{K} P_{a_i}(t) = 1)$. The probability vector $P(t)$ keeps track the probability of being selected for all operators at iteration t.

In this random selection scenario, we assume that $P_{a_i}(t) = \frac{1}{K} (\forall a_i \in A)$. We select the order of the operators randomly at each iteration t. We simply ignore whether operators are successfully applied; therefore, it is not necessary to update the values of $P_{a_i}(t)$ at each iteration.

2.2 Rank-based Selection

In this scenario, the initial value of each operator i at iteration 0 is set as follows: $P_{a_i}(0) = \frac{1}{K} (\forall a_i \in A)$. After all operators are run at one particular iteration, the reward vector $R(t) = \{R_{a_1}(t), \ldots, R_{a_K}(t)\}$ is generated. This reward vector is an important component of the adaptive schemes, whereby operators are rewarded according to their performance. We use a simple way of calculating the reward values. The value of $R_{a_i}(t)$ for operator a_i is a binary value, either 0 (unsuccessful operator) or 1 (successful operator).

In the context of the OP, an operator is successful if it is able to either increase the total remaining travel time (time budget) or improve the objective function value (total collected score). We then calculate the probability of selecting operator a_i for next iteration $(t+1)$, $P_{a_i}(t+1)$, based on either Equation 1 (for a successful operator) or Equation 2 (for an unsuccessful operator).

$$P_{a_i}(t+1) = P_{a_i}(t) + \lambda_1 R_{a_i}(t)(1 - P_{a_i}(t)) - \lambda_2 (1 - R_{a_i}(t)) P_{a_i}(t) \quad (1)$$

$$P_{a_i}(t+1) = P_{a_i}(t) - \lambda_1 R_{a_i}(t) P_{a_i}(t) + \lambda_2 (1 - R_{a_i}(t))[(K-1)^{-1} - P_{a_i}(t)] \quad (2)$$

Algorithm 2 LOCAL SEARCH (RANK-BASED SELECTION)

Generate $\widetilde{A}(t)$ //at iteration t
for all $i \in \widetilde{A}(t)$ **do**
  Call the operator $\tilde{a}_i(t)$
  Determine the value of $R_{\tilde{a}_i}(t)$
  if $R_{\tilde{a}_i}(t) = 0$ **then**
    Update $P_{\tilde{a}_i}(t+1)$ using Equation 1
  else
    Update $P_{\tilde{a}_i}(t+1)$ using Equation 2
  end if
end for

Algorithm 3 LOCAL SEARCH (FITNESS PROPORTIONATE SELECTION)

$AccumProb_0 \leftarrow 0$
for all $a_i \in A$ **do**
  $AccumProb_i \leftarrow AccumProb_{i-1} + P_{a_i}(t)$
end for
$\bar{A} = \emptyset$
repeat
  $U \leftarrow rand(0,1)$
  Find operator a_j such that $AccumProb_{a_{j-1}} < U \leq AccumProb_{a_j}$
  if a_j has not been included in $\bar{A}$ **then**
    $\bar{A} \leftarrow \bar{A} \cup \{a_j\}$
  end if
until all operators are included in $\bar{A}$
for all $i \in \bar{A}(t)$ **do**
  Call the operator $\bar{a}_i(t)$
  Determine the value of $R_{\bar{a}_i}(t)$
  if $R_{\bar{a}_i}(t) = 0$ **then**
    Update $P_{\bar{a}_i}(t+1)$ using Equation 1
  else
    Update $P_{\bar{a}_i}(t+1)$ using Equation 2
  end if
end for

The λ_1 and λ_2 values are the learning rates used to update the selection probabilities. The first one is used to reward an action while the latter parameter is to penalize an unfavorable action.

At iteration t, we generate $\widetilde{A}(t) = \{\tilde{a}_1(t) \geq \ldots \geq \tilde{a}_K(t)\}$ where $P_{\tilde{a}_1}(t) \geq P_{\tilde{a}_2}(t) \geq \ldots \geq P_{\tilde{a}_K}(t)$. The adaptation mechanism that decides which operator to select first is based on the sequence in $\tilde{A}(t)$. The operator with the highest probability value would be selected first. The details are shown in Algorithm 2.

2.3 Fitness Proportionate Selection

The details of fitness proportionate selection (Roulette-Wheel selection) are summarized in Algorithm 3. Similar to the rank-based selection scenario, the initial value of each operator i is set as follows: $P_{a_i}(0) = \frac{1}{K}(\forall a_i \in A)$. The next step is to generate $\bar{A}(t) = \{\bar{a}_1(t), \ldots, \bar{a}_K(t)\}$. The sequence of elements in $\bar{A}(t)$ is defined as follows.

$AccumProb_0$ is initially set to 0. The accumulative of probability values of operator a_i, $AccumProb_{a_i}$ is calculated subsequently by adding $AccumProb_{a_{(i-1)}}$ and $P_{a_i}(t)$.

In order to arrange the order of operators in $\bar{A}(t)$, we generate a random number $U \sim rand(0,1)$ and find operator a_j such that $AccumProb_{a_{(j-1)}} < U \leq AccumProb_{a_j}$. The underlying assumption of the Fitness Proportionate Selection (Roulette-Wheel Selection) is the chance of selecting operator i is proportional to the probability value $P_{\bar{a}_i}(t)$.

This loop is repeated until all operators are inserted in $\bar{A}$. Each operator is called according to its sequence in $\bar{A}$. We then calculate the respective $R(t) \in \{0,1\}$ including $P(t+1)$ value of the selected operator. This is applied until all operators have been called and we move to the next iteration.

3 Computational Experiments

In the first subsection, we provide a short description of the benchmark instances of the TDOP, including the state-of-the-art algorithms for comparison purpose. We then explain how we set up the experiment and analyze the results obtained for the TDOP in the next subsections.

3.1 Benchmark Instances and Approach Comparison

Verbeeck et al. [21] introduced a mathematical model for the TDOP. An issue related to the modeling of time-dependent travel times and the generation of realistic benchmark instances are also discussed. The characteristics of benchmark TDOP instances are summarized in Table 1. All instances are also available in `http://www.mech.kuleuven.be/en/cib/op#section-23`. The number of nodes vary from 21 to 102 nodes. An algorithm based on an Ant Colony System (ACS-TDOP) [21] is the state-of-the-art algorithm for the TDOP. This algorithm obtains high-quality results within very small computational times on benchmark instances.

Table 1: Benchmark TDOP instances

Dataset	Number of instances	Number of nodes
1	9	32
2	9	21
3	9	33
4	10	100
5	3	66
6	9	64
7	10	102

Table 2: Estimation of single-thread performance

Algorithm	Experimental environment	Estimate of single-thread performance
ADILS	Intel(R) Core(TM) with an i5 3.2 GHz CPU and 12 GB RAM	1
ADSAILS	Intel(R) Core(TM) with an i5 3.2 GHz CPU and 12 GB RAM	1
ACS-TDOP	PC with an i5 2.6 GHz CPU and 8 GB RAM	20.64

Table 3: Operators of ILS [7] and SAILS [8]

Operators	Description
INSERT	Insert nodes into a path
SWAP	Swap two nodes within one path
2-OPT	Reorder the sequence of certain nodes within one path
REPLACE	Replace one scheduled node with one unscheduled node

Table 4: Proposed algorithms

Algorithms	Abbreviations
ADILS & ADSAILS with random selection	ADILS-RAND & ADSAILS-RAND
ADILS & ADSAILS with rank-based selection	ADILS-RANK & ADSAILS-RANK
ADILS & ADSAILS with fitness proportionate selection	ADILS-AUTO & ADSAILS-AUTO
ADILS & ADSAILS with fixed sequence	ADILS-FIX & ADSAILS-FIX

3.2 Algorithm Setup

Take note that ADILS, ADSAILS and ACS-TDOP are executed in 5 runs. In order to ensure the fairness among algorithms, we use the *SuperPi* benchmark to adjust the computational time to the speed of the computers used in other solutions [10]. The performance of our machine is set to 1 and other processors are estimated by multiplying with the single-thread performance estimation, as shown in Table 2. For example, if ACS-TDOP is run for 2 seconds, the computational budget using our processor is set to (20.64×2) seconds.

Table 3 summarizes the description of operators used in both ILS [7] and SAILS [8]. The operators for the TDOP are arranged in the following sequence: INSERT $\rightarrow$ SWAP $\rightarrow$ 2-OPT $\rightarrow$ REPLACE since the problem only concern with a single path. These sequences would be treated as the initial sequence in our ADILS and ADSAILS and adjusted accordingly based on our proposed scenarios. Table 4 summarizes our proposed algorithms with their abbreviations that would be used in this paper. The learning rates (λ_1 and λ_2) for LA are set to 0.5 and 0.01, respectively.

3.3 Computational Results

We start this section by comparing three different scenarios for generating the sequence of operators. We also implement the fixed sequence scenario [7, 8] and compare the results. All above-mentioned algorithms are compared with different values of computational times. Since the main objective is to generate solutions within few seconds, we set the computational times (CPU times) within a range of 1 to 10 seconds.

Table 5: The performance of the proposed algorithms on the TDOP instances with respect to the optimal solutions

Algorithm	CPU time = 1 second		CPU time = 3 seconds		CPU time = 5 seconds		CPU time = 10 seconds	
	Best (% gap)	Average (% gap)	Best (% gap)	Average (% gap)	Best (% gap)	Average (% gap)	Best (% gap)	Average (% gap)
AdILS-Rand	2.21	3.99	1.46	2.86	1.22	2.46	0.98	1.84
AdILS-Rank	2.54	4.30	1.56	2.80	1.27	2.40	0.96	1.93
AdILS-Auto	2.25	3.99	1.60	2.99	1.33	2.47	0.98	1.87
AdILS-Fix	2.62	4.01	1.70	2.99	1.46	2.60	1.01	2.12
AdSAILS-Rand	1.93	3.30	1.55	2.82	1.51	2.37	1.11	1.92
AdSAILS-Rank	1.87	3.37	1.51	**2.50**	1.39	2.24	1.03	1.90
AdSAILS-Auto	**1.53**	**3.22**	**1.16**	2.51	**1.10**	**2.21**	**0.94**	**1.83**
AdSAILS-Fix	1.90	3.42	1.61	2.70	1.71	2.53	1.16	1.94

Table 6: The performance of the proposed algorithms on the TDOP instances per dataset

Dataset	Metric	Algorithm						
		AdILS-Rand	AdILS-Rank	AdILS-Auto	AdILS-Fix	AdSAILS-Rand	AdSAILS-Rank	AdSAILS-Auto	AdSAILS-Fix
1	Best(%gap)	**0.00**	**0.00**	0.26	**0.00**	0.53	0.53	0.26	0.83
	Average(%gap)	0.53	**0.37**	0.58	0.41	1.27	1.32	1.18	1.38
2	Best(%gap)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Average(%gap)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
3	Best(%gap)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Average(%gap)	0.19	0.13	0.30	**0.09**	0.39	0.30	0.43	0.53
4	Best(%gap)	4.97	5.90	5.46	6.65	4.13	3.99	**3.60**	3.66
	Average(%gap)	9.07	10.16	9.34	10.33	**7.08**	7.70	7.10	7.34
5	Best(%gap)	4.32	3.87	4.66	5.31	5.25	5.06	**2.64**	3.95
	Average(%gap)	7.45	6.94	7.32	7.06	7.37	7.44	**6.65**	6.65
6	Best(%gap)	1.90	2.68	2.18	1.50	1.28	**0.91**	0.98	1.60
	Average(%gap)	3.85	4.07	3.58	3.00	2.76	**2.47**	2.54	2.66
7	Best(%gap)	5.09	5.51	4.20	5.86	4.04	4.21	**3.52**	4.15
	Average(%gap)	8.13	9.01	7.98	8.03	6.20	6.29	**6.18**	6.71

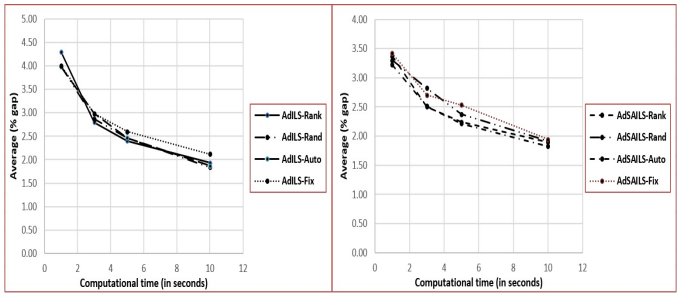

Fig. 2: Algorithms results showing the impacts of computational time

Table 7: The performance of the proposed algorithms on based on ACS-TDOP's CPU times

Algorithm	Best (% gap)	Average (% gap)
ADILS-RAND	0.85	2.05
ADILS-RANK	0.98	1.93
ADILS-AUTO	0.93	1.80
ADILS-FIX	0.91	1.93
ADSAILS-RAND	1.03	2.05
ADSAILS-RANK	1.21	2.03
ADSAILS-AUTO	**0.82**	1.80
ADSAILS-FIX	1.36	1.92
ACS-TDOP	0.94	**1.38**

The performance of algorithms is determined by calculating the percentage deviation from the optimal solution for each benchmark instance. The optimal solutions are obtained by using the commercial solver, CPLEX [21]. We collect the best deviation and average deviation values from 5 runs for each instance. The overall average of best and average deviation values for all instances, BEST (% GAP) and AVERAGE (% GAP), are summarized in Table 5.

Table 5 yields insights into the effect of computational times. In general, ADSAILS performs better than ADILS for all scenarios. BEST (% GAP) values of ADSAILS are below 2% while those of ADILS are above 2% for CPU = 1 second. Similar observations are applied to other computational times. We observe that the results keep improving when the computational time is increased for all algorithms (Figure 2). All algorithms obtain high-quality results on benchmark instances since the values of BEST (% GAP) and AVERAGE (% GAP) are between 0.00% to 4.30%. ADSAILS-AUTO is able to deliver the best performance, except for AVERAGE (% GAP) with CPU = 3 seconds.

We compare the results based on the group of instances by calculating BEST (% GAP) and AVERAGE (% GAP) values per dataset. Table 6 only summarizes the results for CPU = 1 second. For datasets 1,2 and 3 which are considered small instances, all algorithms perform well, especially on Dataset 2. The optimal solutions are obtained for every run. For larger instances, the best two performers are ADSAILS-AUTO and

Table 8: Comparison between ADSAILS-AUTO and ACS-TDOP with the same computational time

Dataset	Metric	Algorithm	
		ACS-TDOP	ADSAILS-AUTO
1	Best (% gap)	**0.00**	**0.00**
	Average (% gap)	**0.17**	0.50
2	Best (% gap)	**0.00**	**0.00**
	Average (% gap)	**0.00**	**0.00**
3	Best (% gap)	**0.00**	**0.00**
	Average (% gap)	0.20	**0.13**
4	Best (% gap)	1.84	**1.38**
	Average (% gap)	**2.76**	3.08
5	Best (% gap)	**1.20**	1.78
	Average (% gap)	**1.40**	4.27
6	Best (% gap)	**0.26**	0.51
	Average (% gap)	**0.78**	1.38
7	Best (% gap)	3.09	**2.47**
	Average (% gap)	**3.94**	4.47

ADSAILS-RANK, as highlighted in **bold**. Similar observations can be obtained for other CPU values.

We further compare the performance of ADSAILS-AUTO with the state-of-the-art algorithm, ACS-TDOP [21]. We set the computational times to the ones by the ACS-TDOP and run our proposed algorithms accordingly. From Table 7, three algorithms, ADILS-RAND, ADILS-AUTO and ADSAILS-AUTO, outperform ACS-TDOP in terms of BEST (% GAP) values. On the other hand, ACS-TDOP produces a better value of AVERAGE (% GAP).

Since ADSAILS-AUTO is the best performer with others, we further compare its results with the ones of ACS-TDOP, as shown in Table 8. We observe that ADSAILS-AUTO is comparable to ACS-TDOP. For small instances (datasets 1-3), both are able to obtain the optimal solutions. For the dataset with the largest number of nodes (dataset 7), ADSAILS-AUTO outperforms ACS-TDOP in terms of the values of BEST (% GAP).

4 Conclusion

This paper presents two metaheuristics, Adaptive Iterated Local Search (ADILS) and Adaptive SAILS (ADSAILS) for solving a variant of the Orienteering Problem (OP), namely the Time Dependent Orienteering Problem (TDOP). In both metaheuristics, Local Search (LS) with a set of operators is embedded for generating possible neighborhood solutions.

We introduce a learning mechanism based on Learning Automata to calculate the probability of selecting the operators for the next iteration. These probabilities are used to determine the sequence of the operators of LS. We propose three different scenarios to determine the sequence of the operators: random selection, rank-based selection and fitness proportionate selection.

The performances of three different scenarios are compared with the one of the fixed sequence of operators and the state-of-the-art algorithms as well. ADSAILS with the fitness proportionate selection is considered as the best performer in solving the benchmark TDOP instances.

Further research could focus on implementing the algorithms for real-life applications. The fast computational time of proposed algorithms enables some interesting applications where it is necessary to update paths. Furthermore, we can consider some additional requirements for the TDOP, such as allowing a fleet of vehicles and including time windows, that can add the practical relevance of the problems. Another possible research direction is how to set the learning rate in the learning automaton. Last but not least, it would be interesting to compare with hyper-heuristics and some automated algorithm configurations, such as ParamILS [11].

Acknowledgements This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

1. Abbaspour, R.A., Samadzadegan, F.: Time-dependent personal tour planning and scheduling in metropolises. Expert Systems with Applications **38**(10), 12,439–12,452 (2011)
2. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237**, 81–117 (2013)
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**, 1695–1724 (2013)
4. Burke, E.K., Gendreau, M., Ochoa, G., Walker, J.D.: Adaptive iterated local search for cross-domain optimisation. In: Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO'11), pp. 1987–1994. Dublin, Ireland (2011)
5. Fomin, F.V., Lingas, A.: Approximation algorithms for time-dependent orienteering. Information Processing Letters **83**, 57–62 (2002)
6. Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T.: Integrating public transportation in personalised electronic tourist guides. Computers and Operations Research **40**(3), 758–774 (2013)
7. Gunawan, A., Lau, H.C., Lu, K.: An iterated local search algorithm for solving the orienteering problem with time windows. In: G. Ochoa, F. Chicano (eds.) Proceedings of the 15th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoStar 2015), *Lecture Notes in Computer Science*, vol. 9026, pp. 61–73. Springer (2015)
8. Gunawan, A., Lau, H.C., Lu, K.: SAILS: hybrid algorithm for the team orienteering problem with time windows. In: Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA 2015), pp. 276–295. Prague, Czech Republic (2015)
9. Gunawan, A., Lau, H.C., Vansteenwegen, P.: Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research pp. 1–18 (2016). DOI 10.1016/j.ejor.2016.04.059
10. Hu, Q., Lim, A.: An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research **232**(2), 276–286 (2014)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36**(1), 267–306 (2009)
12. Krempser, E., Fialho, A., Barbosa, H.J.C.: Adaptive operator selection at the hyper-level. In: C.A.C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (eds.) Proceedings of the 12th International Conference on Parallel Problem Solving From Nature (PPSN 2012), *Lecture Notes in Computer Science*, vol. 7492, pp. 378–387. Springer (2012)
13. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)

14. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic as a general problem solver: an implementation in hyFlex. Journal of Scheduling **16**(3), 291–311 (2013)
15. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: an introduction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
16. Smith, J.E., Fogarty, T.C.: Operator and parameter adaptation in genetic algorithms. Soft Computing **1**, 81–87 (1997)
17. Soria-Alcaraz, J.A., Ochoa, G., Carpio, M., Puga, H.: Evolvability metrics in adaptive operator selection. In: Proceedings of the 16th annual conference on Genetic and Evolutionary Computation (GECCO'14), pp. 1327–1334. Vancouver, Canada (2014)
18. Thierens, D.: Adaptive strategies for operator allocation. Studies in Computational Intelligence **54**, 77–90 (2007)
19. Thierens, D.: Adaptive operator selection for iterated local search. In: T. Stützle, M. Birattari, H.H. Hoos (eds.) Engineering Stochastic Local Search Algorithms, *Lecture Notes in Computer Science*, vol. 5752, pp. 140–144. Springer (2009)
20. Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research **209**(1), 1–10 (2011)
21. Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P.: A fast solution method for the time-dependent orienteering problem. European Journal of Operational Research **236**(2), 419–432 (2014)
22. Walker, J.D., Ochoa, G., Gendreau, M., Burke, E.K.: Vehicle routing and adaptive iterated local search within the HyFlex hyperheuristic framework. In: Y. Hamadi, M. Schoenauer (eds.) Proceedings of the 6th Learning and Intelligent Optimization Conference (LION 2012), *Lecture Notes in Computer Science*, vol. 7219, pp. 265–276. Springer (2012)
23. Yuan, Z., Handoko, S.D., Nguyen, D.T., Lau, H.C.: An empirical study of off-line configuration and on-line adaptation in operator selection. In: P.M. Pardalos, M.G.C. Resende, C. Vogiatzis, J.L. Walteros (eds.) Proceedings of the 8th Learning and Intelligent Optimization Conference (LION 2014), *Lecture Notes in Computer Science*, vol. 8426, pp. 62–76. Springer (2014)